

12-1. What did network builder do in the previous lab? It is instructive to look at the .hoc code because you might want to use this as a template for other simulations.

The first real statement `create acell_home_` is needed because artificial cells need to be created just as real cells do and the default name is `acell_home_`. We then access this to have an accessed section.

Next is the `begintemplate` statement.

We define a cell template so that we can make many copies of a cell to use in network models. This template becomes an “object” like AlphaSynapse or IClamp that we have used before.

The `public` statement allows these procedures and sections to be accessed outside of the template. Normally what is in the template stays in the template! Now we will be able to change parameters of any section if we want to do so later.

Next is `objref synlist`. We will put our synapses in a list. This is the most useful way to organize synapses when there are many of them.

The procedures under `proc init()` are items used to create the cell and should be familiar. The `geom._nseg()` is the procedure that does the `d_lambda` calculation for `nseg`.

`Synlist` is now defined as a new `List` object

The `synapses()` procedure defined later will set up our synapses.

`Init` makes the cell by executing the procedures

Then we create the sections of our cell as usual

The `topol()` procedure establishes the connections of the segments (`connect child, parent`). The `basic_shape()` procedure describes the shape of the cell as we drew it in cell builder. The 3d information provided refers to the positions on the screen and is not related to actual length and diameter. The `pt3dclear()` procedure erases all current 3-D position information and the two `pt3dadd` procedures define the “0” end and the “1” end of the section on the screen. The arguments to these procedures are the x, y, z coordinates and diameter. Because the screen represents the cell in two dimensions the z coordinate is 0 here. Diameter = 1 always here, but this is just the thickness of the line. Can you predict how the cell was drawn from the coordinate information?

(If we build our network outside of the network builder, we may or may not include position information. We might do this to arrange cells in a particular array and we might use the position information to develop “nearest neighbor” connectivity)

12-2. Next we define all subsets. Here the subsets are “all” and “apicals”. All and Apicals are objects that are defined as section lists. All sections will be in the “all” list, but only the apical sections go to the “apicals” list.

Geometry is defined just as we did it in lab.

Lambda_f is defined as external to indicate that it is not a function or procedure defined in this template but is an external .hoc file. The geom._nseg() procedure uses the d_lambda-f procedure to define nseg.

Next the biophysics is defined as we did in lab, although some values may be slightly different.

12-3. The position procedure allows you to move the position of the cell on the screen. Because the coordinates in basic_shape() are given relative to the soma, you can change the position of just the soma and the rest of the cell will follow. Here n3d() returns the number of 3-D points used to define the soma (here 2). These points are moved from their current location x3d, y3d, z3d to a new location specified by the parameters passed \$1, \$2, and \$3. Then the absolute position of the soma is assigned to x, y, and z.

The connect2target procedure sets up the axon so that it can connect to other cells. This procedure will be called outside the template to set up network connections. What this statement does is define an object (\$o2) in the section axon that is a network connection. The voltage at the "1" end of the axon determines if there is input to the target synapse specified by \$o1. Now NetCon can get other parameters (threshold, delay and weight with default 10, 1, 0) but here we accept the defaults. There will be input to the target when the voltage in the axon get above +10 mV and will occur after a synaptic delay of 1 ms. The weight is 0 here and will be set later. The target must be a point process (synapse).

Syn_ is then defined as an object and the procedure synapses defines a syn_ as an Exp2Syn object on each of the 3 apical dendrites and these synapses are appended to the synapse list. The section and location define where the synapse is located. Here the first 2 synapses take the default tau1, tau2 and e values so these are not modified. The last synapse used different defaults so these are changed for this last synapse.

This finishes our template for the pyrcell.

12-4.

Now we set up another template for the artificial excitatory input.

We define some public variables

The artificial cell was created earlier and it is listed as external

In this artificial cell we define a netStim object named pp. NetStim is built into NERURON and generates a train of presynaptic stimuli, so it is a source for NetCon objects. This object takes interval, number, and start as parameters to specify the number of spikes in the train, the spike interval and when they start.

Here the connect2target procedure defines a new network connection named \$o2 (usually named netcon) with source being the pp net stimulator and the target being synapse \$o1.

Position() sets the x, y and z coordinates for this stimulator.

This ends the template for the excitatory network stimulator.

Now we set up an inhibitory network stimulator template in the same way.

Finally we set up the network. Cells, nclist and netcon are defined as objects.

Cells and nclist are defined as lists. Cells will list cells in the network and nclist will be a list of network connections.

Cell_append is a procedure that appends cells to the cell list.

It takes as arguments the cells and their position. This procedure is then executed.

Each cell appended to the list is defined as new. \$o1 is the name of a template cell, \$2 is the x coordinate, \$3 is the y coordinate and 0 is the z coordinate.

The nc_append procedure adds connections. How this is done will depend on whether there are 4 or 5 arguments passed to the procedure,

12-4 – 12.5.

If there are 5 arguments the first (\$1) is the index to the cells list to specify the source cell for the connection, whose connect2target procedure will be used. The source cell connect2target needs a target cell which is obtained from the cell list (\$2) and the index to the synapse list for the particular synapse targeted on this cell (\$3) and the name of the connection (netcon). (See the connect2target procedure on 12-3 which receives \$o2=netcon and \$o1 = the particular synapse on the target cell.) The weight is passed as \$4 and the delay is passed as \$5.

If there are 4 arguments, usually the source is a net stimulator and instead of a synapse as the target we may have the netstim as the target. This is for the case where an event triggers the net stim to start. If the net stim receives a negative weight it shuts off and if it receives a positive weight it starts as configured.

(In newer versions of network builder the syntax here is slightly different. Your .ses file may say If \$3 >= 0 instead where \$3 is skipped with netstim as the target. We will see this later)

In this example we only have 5 arguments passed.

Finally the connection is appended to the connection list.

The nc_append procedure is invoked for the connections we have specified. The parameters passed to the procedure are the cell index for the source cell, the cell index for the target cell, the synapse index on the target cell, the weight of the synapse, and the synaptic delay.

The concepts of how the network builder made the network can be applied to coding networks by hand and we will do this next.

12-6. To code networks by hand we need to

- 1) define all cell types that are going to be used more than once in a template (both real and artificial cells)
- 2) Create all of the cells in the network
- 3) define all current sources, synapses, and artificial input. This can be done separately as shown in the following example or else with the use of lists as in the Network builder .hoc file.
- 4) Define all connections between cells
- 5) Then start nrngui, load the .hoc file with the code for steps 1-4, make graphs & runcontrol.

12-7. Here is an example without the use of lists.

Step 1. We start with a template called Cell which has all of the morphological and biophysical information for our cell.

Step 2. Then we create 2 cells for our network called postcell and precell. We also have to create acell_home for our artificial input

Step 3. We create a current clamp for precell

12-8. and then create synapses. If all cells have the same synapses we could define the synapses in the template. Here we define them individually. These synapses are Exp2Syn type synapses, which as we saw before, are point processes with a NET_RECEIVE block. We could have used a list instead.

Then we create the artificial input by creating 2 NetStim objects with different intervals, number of spikes and starting times.

12-9. Step 4. We create the connections from axons to synapses

The axon from precell connects with syn2 which is on postcell ap[1] at 0.8 and is excitatory
We give this synapse a weight and delay

The axon from postcell connects to syn1 on precell at ap[0] at location 0.1 and is inhibitory

Finally we connect one artificial stimulator to syn3 on precell at ap[2] at 0.1 (excitatory) and we connect the other one to postcell at ap[2] at 0.8 (inhibitory).

So stimulator 1 excites precell and precell excites postcell, but postcell receives inhibition from stimulator 2 and inhibits precell. Meanwhile, precell receives current input to the soma.

12-10. Here is the NetCon description from the documentation which allows you to see the types of arguments that are passed. Default threshold, delay and weight are 10,1, and 0. This means that the voltage must get to +10 mV for a signal to be passed, the signal is delayed by 1 ms and the weight is 0 (meaning you have to set it).

12-11. Here is NetStim from the documentation. Besides interval, number and start parameters that we have already seen, there is a parameter called noise, which can be set to give some variability to the input train.

12-12. Here is an example based more closely on the lab exercise. What was done here was to take the hoc file generated by Network Builder and then modify it to look like what you might write if you were coding this without the Network Builder.

This page looks a lot like 12-1. Differences are that the 3d information has been deleted.

The pyrcell template is started, sections are created, and then the geometry and topology are specified. Then subsets are created for apicals and all.

12-13. Then the biophysics is added as in the lab. This should be familiar to you by now. Then the d_lambda is used to set nseg for each segment.

If the cell is defined with 3d coordinates, it is a good idea to make NEURON calculate the area to make sure the diameters are assigned before you do the nseg. We had an interesting case in my lab once where we tried to use diameter before calculating area and there were problems. This is a quick fix.

Finally we set up the connect2target procedure. NetCon objects are defined from the axon at location 1 to targets.

12-14. Then we have a synapses procedure that defines synapses and adds them to a list. We define 3 synapses on our template cell and provide parameter values as needed.

This ends the cell template.

Then we define an artificial stimulator to deliver excitation and another to deliver inhibition. These are connected to synapses on a target cel.

12-15. This ends our templates. Now we define a list for cells and another list for synapses. Here we define a function that appends cells to a list. We could have used a procedure instead, but without the return statement

We also define a function that appends synapses to a list. Again we could have used a proc

The syntax here matches more recent .hoc files created by the Network Builder. The connections are defined and a weight and delay are assigned.

Now that we have these functions (or procedures) in place we invoke them to create cells, add them to the cell list, and define connections from axons of presynaptic cells to specific synapses on postsynaptic cells with a weight and delay. The first nc_append says the axon of cell 1 (the

artificial stimulator) is connected to cell 0 (P_pyrCell[0]) at synapse 0 (the one on ap[1]) with weight .012 and delay 1).

This .hoc file can be loaded into nrngui. You can then bring up run control and tables for the artificial stimulator parameters and the synapse characteristics from the tools menu items.

However you won't see a panel for synaptic weights or delays. You can change these with

```
nclist.object(2).weight=.008
```

```
nclist.object(2).delay=1
```

Or you could make a loop to change all weights

Or you could code a proc that allows you to pass an index and weight

You also will not see the raster plot that was present in the network builder. You can create one of these in hoc code. For these and other features, see Chapter 11 of The NEURON Book, especially section 11.5.